08-23-00

JC821 U.S. PTO 08/21/00

jc490 U.S. PTO 09/643316 08/21/00

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	2729.200
First Named Inventor or Application Identifier	DAVID E. BROOKLER, et al.
Express Mail Label No.	EL582470228US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: Commissioner for Patents
Box Patent Application
Washington, DC 20231

1. [✓ X] Fee Transmittal Form *(Submit an original, and a duplicate for fee processing)*
2. [✓ X] Specification — *Total Pages* 50
3. [✓ X] Drawing(s) *(35 USC 113)* — *Total Sheets* 11
4. [✓ X] Patent Application Bibliographic Data Sheet — *Total Sheets* 3
5. [ ] Oath or Declaration — *Total Pages* ____
   a. [ ] Newly executed (original or copy)
   b. [ ] Unexecuted for information purposes
   c. [ ] Copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional with Box 18 completed)* ***[Note Box 6 below]***
      i. [ ] DELETION OF INVENTOR(S) — Signed Statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
6. [ ] Incorporation By Reference *(useable if Box 5c is checked)*
   The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 5c, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.
7. [ ] Microfiche Computer Program *(Appendix)*
8. [ ] Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all necessary)*
   a. [ ] Computer Readable Copy
   b. [ ] Paper Copy (identical to computer copy)
   c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

9. [ ] Assignment Papers (cover sheet & document(s))
10. [ ] 37 CFR 3.73(b) Statement *(when there is an assignee)* [ ] Power of Attorney
11. [ ] English Translation Document *(if applicable)*
12. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
13. [ ] Preliminary Amendment
14. [ ] Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
15. [✓ X] Copy of Small Entity Statement [✓ X] Statement filed in prior application Status still proper and desired
16. [ ] Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
17. [ ] Other: ____________________

18. If a CONTINUING APPLICATION, *check appropriate box and supply the requisite information:*
[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No. __/________
Prior application information: *Examiner* ____________ *Group/Art Unit:* ____________

19. CORRESPONDENCE ADDRESS

[✓ X] Customer Number or Bar Code Label 05514 (Insert Customer No. or Attach bar code label here) or [ ] Correspondence address below

NAME					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

+

CLAIMS	(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) CALCULATIONS
	TOTAL CLAIMS (37 CFR 1.16(c))	63 -20 =	43	X $ 18.00 =	$ 774.00
	INDEPENDENT CLAIMS (37 CFR 1.16(b))	6 -3 =	3	X $ 78.00 =	$ 234.00
	MULTIPLE DEPENDENT CLAIMS (if applicable) (37 CFR 1.16(d))			$260.00 =	- 0 -
				BASIC FEE (37 CFR 1.16(a))	$ 690.00
			Total of above Calculations =		$1,698.00
	Reduction by 50% for filing by small entity (Note 37 CFR 1.9, 1.27, 1.28).				$ 849.00
				TOTAL =	$ 849.00

20. Small entity status
    - a. [ ] A small entity statement is enclosed
    - b. [X] A small entity statement was filed in the prior nonprovisional application and such status is still proper and desired (copy enclosed).
    - c. [ ] Is no longer claimed.

21. [X] A check in the amount of $ 849.00 to cover the filing fee is enclosed.

22. [ ] A check in the amount of $ ________ to cover the recordal fee is enclosed.

23. The Commissioner is hereby authorized to credit overpayments or charge the following fees to Deposit Account No. 06-1205:
    - a. [X] Fees required under 37 CFR 1.16.
    - b. [X] Fees required under 37 CFR 1.17.
    - c. [ ] Fees required under 37 CFR 1.18.

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT REQUIRED	
NAME	Carole A. Quinn, Reg. No. 39,000
SIGNATURE	Carole A. Quinn
DATE	August 21, 2000

CA_MAIN 7817 v 1

"Express Mail" mailing label number EL583470228US

Date of Deposit August 21, 2000

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D C. 20231

Carole A. Quinn

(Typed or printed name of person mailing paper or fee)

Carole A. Quinn

(Signature of person mailing paper or fee)

STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & 1.27(c))--SMALL BUSINESS CONCERN

Docket Number (Optional) 9407-0F861US0

Applicant, Patentee, or Identifier: David E. Brookler, et al
Application or Patent No.:
Filed or Issued: August , 1999
Title: Method and System for Efficient Structuring, Storing and Searching of Data in an RDBMS

I hereby state that I am:
- [ ] the owner of the small business concern identified below:
- [✓] an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF SMALL BUSINESS CONCERN A2i, Inc.

ADDRESS OF SMALL BUSINESS CONCERN 1925 Century Park East, Suite 255, Los Angles, California 90067-2703

I hereby state that the above identified small business concern qualifies as a small business concern as defined in 13 CFR Part 121 for purposes of paying reduced fees to the United States Patent and Trademark Office. Questions related to size standards for a small business concern may be directed to: Small Business Administration, Size Standards Staff, 409 Third Street, SW, Washington, DC 20416.

I hereby state that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention described in:

- [✓] the specification filed herewith with title as listed above.
- [ ] the application identified above.
- [ ] the patent identified above.

If the rights held by the above identified small business concern are not exclusive, each individual, concern, or organization having rights in the invention must file separate statements as to their status as small entities, and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d), or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern, or organization having any rights in the invention is listed below:
- [✓] no such person, concern, or organization exists.
- [ ] each such person, concern, or organization is listed below.

Separate statements are required from each named person, concern or organization having rights to the invention stating their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

NAME OF PERSON SIGNING Paul N. Weinberg

TITLE OF PERSON IF OTHER THAN OWNER Vice President

ADDRESS OF PERSON SIGNING 2160 Century Park East, #1905, L.A., CA, 90067

SIGNATURE [illegible] DATE 8/17/99

INVENTOR INFORMATION

Inventor One Given Name:: David E.
Family Name:: Brookler
Postal Address Line One:: 1700 South Shenandoah Street
City:: Los Angeles
State or Province:: California
Country:: United States
Postal or Zip Code:: 90035
City of Residence:: Los Angeles
State or Province of Residence:: California
Country of Residence:: United States
Citizenship Country:: United States

Inventor Two Given Name:: Ariel
Family Name:: Hazi
Postal Address Line One:: 11963 Victoria Avenue
City:: Los Angeles
State or Province:: California
Country:: United States
Postal or Zip Code:: 90066
City of Residence:: Los Angeles
State or Province of Residence:: California
Country of Residence:: United States
Citizenship Country:: United States

Inventor Three Given Name:: Dave L.
Family Name:: Sullivan
Postal Address Line One:: 5055 Bakman Avenue
Postal Address Line Two:: Apartment 101
City:: North Hollywood
State or Province:: California
Country:: United States
Postal or Zip Code:: 91601
City of Residence:: North Hollywood
State or Province of Residence:: California
Country of Residence:: United States
Citizenship Country:: United States

"Express Mail" mailing label number EL582470228US
Date of Deposit August 21, 2000
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231.

Carole A. Quinn
(Typed or printed name of person mailing paper or fee)

[illegible]
(Signature of person mailing paper or fee)

Inventor Four Given Name:: Dominic
Family Name:: Tham
Postal Address Line One:: 3700 Bagley Avenue
Postal Address Line Two:: Apartment 115
City:: Los Angeles
State or Province:: California

Country:: United States
Postal or Zip Code:: 90034
City of Residence:: Los Angeles
State or Province of Residence:: California
Country of Residence:: United States
Citizenship Country:: United States

Inventor Five Given Name:: Philip A.
Family Name:: Tinari
Postal Address Line One:: 9649 Olympic Boulevard
Postal Address Line Two:: Apartment 5A
City:: Beverly Hills
State or Province:: California
Country:: United States
Postal or Zip Code:: 90212
City of Residence:: Beverly Hills
State or Province of Residence:: California
Country of Residence:: United States
Citizenship Country:: United States

Inventor Six Given Name:: Paul N.
Family Name:: Weinberg
Postal Address Line One:: 2160 Century Park East
Postal Address Line Two:: Apartment 1905
City:: Los Angeles
State or Province:: California
Country:: United States
Postal or Zip Code:: 90067
City of Residence:: Los Angeles
State or Province of Residence:: California
Country of Residence:: United States
Citizenship Country:: United States

CORRESPONDENCE INFORMATION

Correspondence Customer Number:: 05514
Fax:: (212) 218-2200

APPLICATION INFORMATION

Title Line One:: DATA INDEXING USING BIT VECTORS

Total Drawing Sheets:: 11
Formal Drawings?:: Yes
Application Type:: Utility
Docket Number:: 2729.200
Secrecy Order in Parent Appl.?:: No

REPRESENTATIVE INFORMATION

Representative Customer Number:: 5514

PRIOR FOREIGN APPLICATIONS

Priority Claimed:: No

CA_MAIN 7909 v 1

"Express Mail" mailing label number EL582490228US

Date of Deposit August 21, 2000

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231.

Carol A. Quinn

(Typed or printed name of person mailing paper or fee)

[illegible]

(Signature of person mailing paper or fee)

DATA INDEXING USING BIT VECTORS

This application claims the benefit of U.S. Provisional Application No. 60/149,855, filed August 19, 2000.

BACKGROUND OF THE INVENTION

Field Of The Invention

The present invention relates to indexing data and, more particularly, to a method and apparatus wherein bit vector indexing is used to index data such as record data in a database.

Description Of The Related Art

A DBMS (Database Management System) is used to manage data and is comprised of computer-executable code that may be used to define the structure of data and to access data within the defined structure. One example of a DBMS is a relational DBMS, or RDBMS. An RDBMS manages tables that make up a relational database as well as the data contained in the tables. In an RDBMS, data is organized in rows (or records) and columns (or fields) of the tables, and two or more tables may be related based on like data values. The intersection of a row and column in a table is referred to as a cell and contains the data value for a particular field of a particular record.

A DML (data manipulation language) such as SQL (Structured Query Language) is typically used to store, retrieve and modify data in a table. A schema defines the structure of a database, i.e., each table and the fields within a record of a table. A schema is itself considered data that is stored in one or more tables. Therefore, like other data in a database, a DML may be used to store, retrieve and modify the data in the database as well as the structure of a database.

There are performance issues with respect to data access in a DBMS particularly when the database is very large. A typical RDBMS is optimized for certain types of query access. However, performance degrades when the database is very large, when a query returns a large set of records, when row selection criteria apply across multiple fields and tables, or when interactively browsing large sets of query results. Value limiting on a lookup table reduces the set of lookup values by eliminating from the set of all possible lookup values those values that do not correspond to any records in the primary table. Another problem is that value limiting cannot efficiently and quickly be done using the standard mechanisms of an RDBMS.

An RDBMS uses indexes to assist in performing queries and quickly locate records in the database. Indexes store one or more field (or column) values from each record as a unique key for the record. Indexes do an adequate job of speeding up the query process even on large databases when the row selection criteria include constraints on only a single field and when the query results do not need to be browsed interactively. In particular, an RDBMS can quickly search for and retrieve an individual record from among even millions of records based on the value of an indexed field.

Unfortunately, however, there are a number of shortcomings to searching using conventional forms of indexing.

For example, if the number of records in the database is very large, the index itself can become large as well, so that as a practical matter it will be stored on disk rather than in memory, moderately increasing the time necessary to search for records.

Further, if the row selection criteria includes constraints for multiple fields, the speedup only applies to each field individually. The problem of then reconciling the multiple sets of query results, one for each constraint, into a single set of query results for all the constraints, requires complex algorithms and heuristics that can dramatically increase the time necessary to execute the query. In fact, the time required grows geometrically with the number of records in the individual result sets, which means that query constraints that return very large sets of records reduce system performance.

A further disadvantage of conventional indexing relates to situations in which a user wishes to interactively browse query results which results in performance degradation. Most relational database management systems support interactive browsing using cursors and temporary files, which require that the entire set of query results first be written to disk before browsing, a very slow operation compared to memory access. Moreover, the set of query results must be accessed and written to disk in its entirety even if only a small subset of the records will ever be brought into view. Again, if the set of query results is very large, the operation of writing them to disk can take a long time.

Also, if a user chooses to build up a query interactively and iteratively by adding one constraint at a time and viewing intermediate query results along the way, the entire query process needs to be repeated from scratch as each additional constraint is added to the query, each time incurring all of the overhead of each of the steps of reading the index, applying multiple row selection criteria, reconciling query results, and writing them to disk.

A final problem arises when attempting to perform value limiting. Value limiting allows the system to present the user with lists of values for search selections that always correspond to records in the primary table, preventing the user from making search selections that lead to no records found. Unfortunately, a typical RDBMS can only accomplish this process through complex, multi-table joins (i.e., a join combines information from two tables by performing a lookup on every record of the primary table) that cannot usually be done quickly enough to provide an acceptable response time in an interactive environment. A lookup uses a pair of matching columns from two tables, taking the value of the column for a single record in the first primary table to "look up" additional information in a single corresponding record in the second lookup table. As a result, value limiting is impractical when performing interactive and iterative searches because the value limiting would have to be done across multiple lookup tables again and again.

Thus, it would be beneficial to have a mechanism to more efficiently index data in data records such as those stored in a database.

SUMMARY OF THE INVENTION

The present invention addresses the foregoing problems and provides for an indexing scheme for indexing data in data records using bit vectors.

In one aspect of the invention, indexing of occurrences of a value in at least one data record using a bit vector representation is provided wherein the bit vector representation is associated with the value and a bit of the bit vector representation is associated with each of the at least one data record, a determination is made whether the value exists in the at least one data record, a bit value is assigned to the bit in the bit vector representation based on the outcome of the determination.

Indexing using BVs (bit vectors) provide several advantages over conventional indexing approaches. Since, a BV uses one bit per record in the primary table instead of a minimum of eight bytes per record for an index, a bit vector is substantially smaller than a conventional index. This results in faster processing and less memory usage, since it is not likely to need to be stored on disk, and if it is, requires that less data be accessed on the disk for a particular operation. Further, it is possible to encode a BV, using any of the well known compression schemes, to further reduce the amount of storage they require.

Further, instead of complex algorithms reconciling individual sets of query results to combine the multiple constraints, logical operations may be used. Unlike the geometric time required to reconcile individual result sets in the conventional searching approaches, the time grows linearly with the number of records in the primary table.

The set of records that satisfy a query correspond to the bit vector that results from bit-wise operations (e.g., "ORs" and "ANDs"). There is no need to create a temporary file of query results and the records themselves do not need to be accessed in advance. In an interactive environment, a particular record need only be accessed when it is browsed into view, if ever.

In addition, BVs reduce the repeated overhead when performing interactive, iterative queries. Intermediate resulting bit vectors can be stored for each lookup field during the course of an iterative query. Additional constraints can then be applied to them rather than reapplying all of the constraints from scratch using the original BVs of the BVIs (Bit Vector Indexes).

Further, operations may be performed on multiple bit vectors to determine the existence of combinations and associations between the corresponding values used in the indexed data records. BVIs are perfectly suited for value limiting across multiple lookup tables and completely eliminate the need to perform complex multi-table joins.

In another aspect of the invention, combinations of values used in at least one data record are identified by creating a first bit vector representation for a first value, the first bit vector representation identifying use of the first value in the at least one data record, creating a second bit vector representation for a second value, the second bit vector representation identifying use of the second value in the at least one data record, and performing a bit-level operation on the first and second bit vector representations.

This brief summary has been provided so that the nature of the invention may be understood quickly. A more complete understanding of the invention can be obtained by reference to the following detailed description of the preferred embodiment(s) thereof in connection with the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is an outward view of a hardware environment embodying the present invention.

Fig. 2 is a block diagram of the internal architecture of a typical computer for use in conjunction with the present invention.

Fig. 3 provides an example of a table and the use of BVIs to index data contained in the table according to the present invention.

Fig. 4 further illustrates the BVs (bit vectors) depicted in Fig. 3.

Fig. 5 provides a diagram of process steps to create a BVIs and BVs according to the present invention.

Fig. 6 provides a diagram of process steps to perform a bit-wise operation on BVs according to the present invention.

Figs. 7A and 7B provide an example of bit-wise "OR" and "AND" operations performed on BVs according to the present invention.

Fig. 8 provides a diagram of process steps to perform value-limiting according to the present invention.

Figs. 9A and 9B provide an example of value-limiting processing performed according to the present invention.

Fig. 10 provides an example of Category-Manufacturer and Manufacturer-Category BVIs according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Fig. 1 is an outward view of representative computing hardware embodying the present invention. Shown in Fig. 1 are computer 10 executing an operating system, display monitor 11 for displaying text and images to a user, keyboard 14 for entering text and commands into computer 10, and mouse 12 for manipulating and for selecting objects displayed on display monitor 11, or for output to an output device such as printer 16. Also included with computer 10 are fixed disk drive 6, in which are stored application programs, such as a DBMS and other applications, data files, and device drivers for controlling peripheral devices attached to computer 10, floppy disk drive 15 for use in reading data from and writing data to floppy disks inserted therein. Data and/or applications may also be accessed from a CD-ROM via a CD-ROM drive (not shown) or over a network to which computer 10 may be connected (network connection not shown).

Fig. 2 is a block diagram of the internal architecture of computer 10. Shown in Fig. 2 are CPU 20, which may be any microprocessor including, but not limited to, a Pentium-type microprocessor, interfaced to computer bus 21. Also interfaced to computer bus 21 are printer interface 25, to allow computer 10 to communicate with printer 16, modem interface 29 to enable communications between computer 10 and a modem, display interface 27 for interfacing with display monitor 11, keyboard interface 28 for interfacing with keyboard 14, mouse interface 23 for interfacing with mouse 12, and network interface 26 for connecting to a network (e.g., Internet, intranet, local area network, etc.).

Read only memory (ROM) 24 stores invariant computer-executable process steps for basic system functions such as basic I/O, start up, or reception of keystrokes from keyboard 14.

Main random access memory (RAM) 30 provides CPU 20 with memory storage which can be accessed quickly. In this regard, computer-executable process steps of a DBMS or other application are transferred from disk 6 over computer bus 21 to RAM 32 and executed therefrom by CPU 20.

Also shown in Fig. 2 is disk 6 which, as described above, includes a windowing operating system, a DBMS which includes data stored therein as well as schema and data stored in one or more tables defined in the schema. Further, disk 6 may be used to store executable-code (e.g., stored procedures) comprising steps described herein indexing data using bit vector representations. Disk 6 further includes data files and device drivers as shown.

The present invention uses a BVI (Bit Vector Index) to augment the standard indexing scheme of a typical RDBMS. A BVI is a collection of BV representations that together comprise an index for a particular column in a table of a database. One example of a BV representation is a bit vector (BV) comprising a sequence of Boolean values, each stored as a single bit. The column in the indexed table stores a reference to a value in a set of enumerated values such as those values found in a lookup table. A lookup mechanism uses a pair of matching columns from two tables, taking the value of the column for a single record in the first table to "look up" additional information in a single corresponding record in the second (lookup) table.

The present invention creates a BVI for each matching column pair that relates a lookup field in the indexed table to a set of values in a lookup table. BVI indexing of the present invention may be used in place of or as a supplement to other forms of indexing.

BVIs and BVs have a number of advantages. Since, a BV uses one bit per record in the indexed table instead of a minimum of eight bytes per record for an index, a bit vector is substantially smaller than a conventional index. This results in faster processing and less memory usage, since a BVI is not likely to need to be stored on disk, and if it is, requires that less data be accessed on the disk for a particular operation. Further, it is possible to encode a BV to generate another BV representation that optimizes the space need for a BV. Further, it is possible to compress a BV representation (optimized or not), using any of the well known compression schemes, to further reduce the amount of storage required for a BVI.

Further, instead of complex algorithms reconciling individual sets of query results to combine the multiple constraints, logical operations may be used. Unlike the geometric time required to reconcile individual result sets in the conventional searching approaches, the time grows linearly with the number of records in the primary table.

In the present invention, a set of records that satisfy a query can be represented as a BV that is the result of bit-wise operations (e.g., "ORs" and "ANDs"). There is no need to create a temporary file of query results and the records themselves do not need to be accessed in advance. In an interactive environment, a particular record need only be accessed when it is browsed into view, if ever.

In addition, BVIs reduce the repeated overhead when performing interactive, iterative queries. Intermediate resulting bit vectors can be stored for each lookup field during the course of an iterative query. Additional constraints can then be applied to them rather than reapplying all of the constraints from scratch using the original BVs of the BVIs.

BVIs are perfectly suited for value limiting across multiple lookup tables and completely eliminate the need to perform complex multi-table joins.

Fig. 3 provides an example of a table and the use of BVIs to index data contained in the table according to the present invention. The particular tables depicted in Fig. 3 are by way of example only, and it should be apparent that the present invention is not limited to this example.

Products table 300 contains four fields: product ID, description, manufacturer and category fields. Each of records 310 through 314 contain information associated with a particular manufacturer and category. The manufacturer and category fields are lookup fields such that the value in each cell stores a reference (or identifier, ID) to a value in a lookup table.

Lookup tables 301 and 302 associate the reference made in records 310 through 314 to an actual value. Table 301 contains values for manufacturers while table 302 contains category values. Referring to table 301, for example, an ID value of "2" corresponds to the manufacturer, "APEX". In products table 300, records 312 and 315 have a value of "2" in the manufacturer field. In a DBMS, the value in the manufacturer field of record 312, for example, may be used to lookup the name of the manufacturer associated with a lookup value of "2". Similarly, a value in the category fields of records 310 through 314 may be used to identify a corresponding category name stored in table 302.

BVIs 303 and 304 contain BVs associated with the manufacturer table 301 and category table 302 (respectively).

A BVI may comprise multiple BVs. Preferably, a BVI such as BVIs 303 and 304 are array structures with each entry containing a BV and the indexed value being a pointer into the array. Further, a BVI is preferably a unidimensional array. However, a BVI may be multidimensional arrays as well. BVIs 303 and 304 of Fig. 3 are depicted as two dimensional arrays that includes the indexed value to further illustrate the functionality of the present invention.

Each BV of a BVI identifies the records in the indexed table that correspond to one particular value in the lookup table. A bit is set in a particular position in the BV that corresponds to a given value, if the corresponding record in the indexed table has that value. The collection of BVs for all of the values of the matching column in the lookup table comprises the BVI for that matching column pair.

Thus, for example, records 330 through 332 of BVIs 303 contain BVs that provide an index of records 310 through 314 and the manufacturers identified in the records. Similarly, records 340 through 342 contain BVs for indexing records 310 through 314 by category.

Fig. 4 further illustrates the BVs depicted in Fig. 3. BVs 430 through 432 provide an index of manufacturers while BVs 440 through 442 index categories. To further illustrate, BV 430 contains five bits each corresponding to a record 310 through 314 of table 300. The leftmost bit corresponds to record 310, the next to record 311, the third to record 312, the fourth to record 313 and the rightmost bit to record 314.

In this example, a value of "1" in a manufacturer's BV indicates that the corresponding record in table 300 references the manufacturer that corresponds to the BV in its manufacturer field.

For example, referring to BV 430, which indexes a manufacturer value of "ACME", bits one and two are "on" indicating that the corresponding records 310 and 312 of table 300 reference the "ACME" manufacturer. Similarly, in BV 441, which indexes the "Computer" category, the third and fifth bits correspond to records 312 and 314 (respectively) and are set to indicate that these records reference the "Computer" category.

A BV contains one bit per record of the indexed table rather than a minimum of eight bytes per record for a conventional index. Thus, a BV is substantially smaller than a corresponding index. A BVI can, therefore, be processed faster and requires less memory. Further, since it requires less storage space, a BVI is not as likely to need to be stored on disk as a conventional index, but if it is, the reduced size of a BVI results in less data being accessed on the disk for a particular operation.

In addition, various encoding techniques may be used to further reduce the size of a BVI and the BVs contained therein. It is possible to encode sparse BVs to further reduce the amount of storage they require. Various encoding schemes that are used include enumeration, run-length encoding, truncation of leading and trailing zeros, and LZW compression, as well as additional compression over the entire BVI.

Compression is especially useful where there are contiguous portions of a BV with like values (i.e., "0"s or "1"s). However, even in the case that a BV contains randomly distributed 1's or 0's, it may be possible to compact the BV by storing bits up to the last bit set.

The present invention provides various encoding mechanisms to optimize storage based on the nature of the BVs. A flag may be associated with each BVI to identify the encoding scheme used for a BVI.

One such BVI structure may be used where there are contiguous portions of a BV, the BV may be stored as an ordered list of either set (e.g., values of "1") or unset bits (e.g., values of "0"). That is, instead of storing a BV as a series of "1"s and "0"s, bit position information is stored. For example, where the first ninety-nine bits are "0" and the hundredth bit is a "1", the first entry in the BV is the value "100". Conversely, where the BV contains fewer "0"s than "1"s, for example, the ordered list may indicate the position of the "0"s in the BV. Such encoding is especially useful where the number of bits set (or unset) is less than 1 in each "n" bits where "n" is the word size (or number of bits used to store a number). For example, such encoding is well suited for BVs where the number of bits set (or unset) is 1 bit in 32 bits for 4-byte numbers, 1 in 16 for 2-byte numbers, 1 in 8 for 1-byte numbers, etc.

A BV may also be stored as ordered start and end pairs of set (or unset) bits. That is, a run of set (or unset) bits are stored as start and stop positions within a BV in which the run begins and ends (respectively). Such encoding is especially useful when the average run, or region size, is greater than twice the word size used to store a number, or a region size of 64 for 4-byte numbers, 32 for 2-byte numbers, etc.

Compression can also be applied to any of the above encoding approaches. However, there are tradeoffs between speed and storage. Compression can act to decrease speed, but is useful where it improves storage. The determination may be made based on the available computer system resources.

As with standard indexes, there is a mapping between a bit of the BV and a record in the indexed table. Preferably, the pointer into a BV is a field in the indexed table (e.g., a record ID field such as the product ID field in table 300) whose value identifies the corresponding bit position in the BV. Alternatively, a mapping mechanism may be used to translate between a record ID and a position in a BV.

A BVI must be synchronized with the fields that it indexes. For an existing table, the values contained in the indexed fields are used to generate a BVI. Thereafter, the BVI is updated whenever a value in the indexed field is modified.

Fig. 5 provides a diagram of process steps to create a BVI and BVs contained therein according to the present invention. The process steps of Fig. 5 are performed for each value in a lookup table (e.g., values 1 through 3 in table 301 or 302) to generate a BVI where a BV in the BVI corresponds to a value in the lookup table.

At step S500, a determination is made whether all of the enumerated values in the lookup table have been processed. If so, processing ends at step 506. If not, processing continues at step S501 to generate a BV for the next enumerated value.

At step S501, the BV for the next enumerated value is initialized. At step S502, a determination is made whether all of the records in the indexed table have been processed for the current enumerated value. If so, processing continues at step S500 to process any remaining enumerated values. If not, processing continues at step S503 to get the enumerated value used in the next record in the indexed table (e.g., one of records 310 through 314 of products table 300).

At step S504, a determination is made whether the value used in the record is the same as the indexed value currently being processed. If not, processing continues at step S502 to process any remaining records in the indexed table. If so, processing continues at step S505 to set the bit, that corresponds to the current record, in the BV for the indexed value currently being processed.

Processing continues at step S502 to process any remaining records in the indexed table.

Searching lookup fields based on lookup values is dramatically faster using a BVI than a traditional index. To identify the set of records in the indexed table that correspond to a particular value in the lookup table, the BV for that value is extracted from the BVI for the lookup table. The bits that are set in the BV immediately identify the set of records. Using this approach, the time required to identify the set of records having a particular value in a lookup field grows linearly rather than geometrically (as in conventional approaches) with the number of records, as well as linearly rather than exponentially (as in conventional approaches) with the number of tables.

In addition, multiple constraints on a single lookup field may be handled using logical operations on BVs. Thus, for example, selecting records based on multiple lookup table values is accomplished using an "OR" operation. Such an operation may be used to select records from products table 300 where the manufacturer is either "ACME" or "Apex". The BVs that correspond to each of the desired lookup table values are "ORed" together. Any bit that is set in the resulting BV indicates that the corresponding record in the indexed table is included in the result set.

Fig. 6 provides a diagram of process steps to perform a bit-wise operation on BVs according to the present invention. At step S601, a result BV is initialized to the first BV to be used in the operation. At step S602, a determination is made whether all operations have been performed. If so, processing ends at step S605. If not, processing continues at step S603, to get the next BV. At step S604, the result BV is updated by performing a bit-wise operation (e.g., an "OR" or "AND" operation) on the result BV and the BV retrieved in step S603. Processing then continues at step S602 to process any remaining BVs and or BVIs.

Operations performed on BVs may proceed hierarchically in addition to the linear approach taken in Fig. 6. That is, for example, operations performed on like BVs are performed first using an "OR" bit-wise operation and bit-wise "AND" operations on dissimilar BVs are performed on the result(s). To illustrate, consider a selection criteria that comprises "Printer" or "Monitor" from either "ACME" or "Apex". Since the first two are both categories of products and the latter two are both manufacturers, a bit-wise "OR" operation is performed to generate a BV that satisfies the category criteria, a bit-wise "OR" operation is performed to generate a BV that satisfies the manufacturer criteria, and the two generated BVs are then bit-wise "ANDed".

Figs. 7A and 7B provide an example of bit-wise "OR" and "AND" operations performed on BVs within BVIs according to the present invention.

Referring to Fig. 7A, an "OR" operation is performed on the "ACME" and "Apex" BVs (from records 330 and 331 of table 303) to identify the records in products table 300 where the manufacturer is either "ACME" or "Apex".

BV 700 from record 330 is "ORed" with BV 701 of record 331 to yield BV 702. Since a value of "1" is used to indicate that the corresponding record in the products table 300 contains the lookup table value (i.e., either "ACME" or "Apex"), the presence of a "1" in BV 702 indicates that either "ACME" or "Apex" appears in the corresponding record. That is, records 310, 311, 312, and 314 contain a manufacturer's value of either "ACME" or "Apex".

BVIs also facilitate searching that involve constraints on multiple fields. In a conventional approach, complex algorithms are used, and reconciling individual sets of query results must be performed to combine the multiple constraints. Instead, the present invention contemplates one or more operations on a set of BVIs that is faster and less complex. After the bit vectors for multiple values constraining a single lookup field are first bit-wise "ORed", the resulting BVs for each of the lookup fields are then bit-wise "ANDed". Unlike the geometric time required to reconcile individual result sets, the time grows linearly with the number of records in the indexed table.

For example, a request for a manufacturer equal to either "ACME" or "Apex" and a category of "Computer" may be satisfied using logical bit-wise operations (e.g. "AND" and "OR") on the corresponding manufacturer and category BVIs.

The appropriate manufacturer BVs are bit-wise "ORed" together as illustrated in Fig. 7A. The result, BV 702, is then bit-wise "ANDed" with the BV corresponding to the "Computer" category. Referring to Fig. 7B, BV 702 is bit-wise "ANDed" with BV 711 (from record 341) to yield BV 712. As indicated by BV 712, records 311 and 312 contain the requested "ACME" and "Apex" computer products.

In the above operations, there was no need to store query (e.g., intermediate or final) results in temporary storage (e.g., file or memory). Further, the appropriate records may be found without the need to access the records themselves. The records that correspond to the set values in the result BV values are the appropriate records. Given records must be accessed only when it is necessary to provide the information contained in the record such as when the record is to be displayed in an interactive environment (e.g., browsing environment). Since the time needed to retrieve all of the records to perform the necessary queries as done in the past is eliminated, the setup time needed to generate the interactive display may be reduced. Records are only accessed when they are to be browsed or otherwise viewed in the interactive environment.

It is also possible to store intermediate, result BVs (e.g., BVs 702 and 712) in a case that a constraint is frequently used, for example. It is then possible to apply additional constraints to the stored, intermediate BVs rather than reapplying all of the constraints on the original BVs thereby reducing the overhead. This is especially useful in a case of incremental queries performed interactively (e.g., by a user or users of an interactive system). It can be seen that storage of intermediate results in the form of intermediate BVs requires much less space than storing intermediate results (i.e., tables containing records) that would be necessary when using a conventional approach.

BVs are perfectly suited for value limiting across multiple lookup tables and completely eliminate the need to perform complex multi-table joins.

BVs may also be used to perform value limiting on a particular lookup field. Value limiting is typically used to limit selection to only those values that correspond to records in the indexed table that satisfy any constraints.

Briefly, value limiting is performed using the present invention by ignoring the constraints on a lookup field being value-limited, so that the next incremental query can change the constraints on a particular lookup field based on all the values for which records exist in the indexed table, not just the values already selected based on constraints on the value-limited lookup field.

If one does not already exist, an intermediate BV is generated using the constraints on any other lookup fields.

A logical "AND" operation is then performed on the intermediate BV and each BV in the BVI for the value-limited lookup field. In contrast to a bit-wise "AND" operation, a logical "AND" returns a single value (i.e., either "TRUE" or "FALSE"). Any value for which the result of the logical "AND" is "FALSE" may be eliminated from the value-limited list. Note that it is not always necessary that all the bits in the BV be compared. That is, the comparison can stop as soon as one pair of corresponding bits are found to both be set.

Fig. 8 provides a diagram of process steps to perform value-limiting according to the present invention. Initially, the constraints on the value-limited field are ignored and, at step S801, the set of valid values for the value-limited field are all the values.

At step S802, a determination is made whether constraints on other lookup value fields have been processed. If not, processing continues at step S803 to generate or update a limiting BV using the next constraint. The limiting BV is used to limit the value-limited field by applying it to each of the BVs in the value-limited field's BVI.

Step S803 may be performed as discussed above with reference to Fig. 6. Processing continues at step S802 to process any remaining constraints on other lookup value fields.

If it is determined, at step S802, that all other constraints have been processed, processing continues at step S804 to determine whether all values in the value list initially created in step S801 have been processed. If so, processing ends at step S805. If not, processing continues at step S806, to get the BV that corresponds to the next value in the value list. At step S807, a logical "AND" operation is performed on the value BV and the limiting BV generated in step S803.

At step S808, the result of the "AND" operation performed at step S807 is examined to determine whether is has a value of “TRUE” or “FALSE”. If the result is “TRUE”, processing continues at step S804 to process any BVs remaining for the value-limited value. If the result is “FALSE”, processing continues at step S809 to remove the value that corresponds to the BV retrieved in step S806 from the value list, and processing continues at step S804.

Figs. 9A and 9B provide an example of value-limiting processing performed according to the present invention. Fig. 9A provides an example of value-limiting manufacturers to those that provide computer products. Initially, a value list 900 is generated that ignores any constraints on manufacturers. That is, value list 900 includes "ACME", "Apex" and "Best".

A result BV is generated which, in the case of the example, is simply the BV for "Computers", or BV 941 (i.e., "01100"). BV 941 is logical "ANDed" with each of BVs 930, 931 and 932 from the manufacturer BVI table 303. BV 930 (for manufacturer ID="1" or "ACME") is logical "ANDed" with BV 941 to yield BV 951. As illustrated by BV 951, the result of a logical "AND" is "TRUE", since both BVs 930 and 941 have the second bit set. Similarly, a logical "AND" between BV 931 (for manufacturer ID="2" or "Apex") and BV 941 results in "TRUE". However, BV 932 (i.e., for manufacturer ID= "3" or "Best") logically "ANDed" with BV 941 yields no set bits (as illustrated by BV 953). Therefore, the result of the logical "AND" is "FALSE". As indicated in value list 901, the value limited set of manufacturers is "ACME" (ID="1") and "Apex" (ID="2"). A validation check of the data will reveal that the only manufacturers with computer products are "ACME" and "Apex".

Similarly, Fig. 9B illustrates value-limiting performed on the category lookup field. To find the valid categories for the "ACME" manufacturer, for example, a result BV is generated while ignoring the Category constraints. In this case, this is simply BV 930 (i.e., "11000") from the manufacturers BVI. BV 930 is logical "ANDed" with each of the category BVs 940, 941 and 942. As illustrated by BVs 954, 955 and 956, logical "AND" operations yield a "TRUE" for ID="1" and ID="2", but a "FALSE" for ID="3". Thus, as indicated in value list 901, the valid categories are "Printers" and "Computers".

Like BVIs 303 and 304 that store BVs for an indexed value relative to data records, it is also possible to create value-limiting BVIs that store BVs to identify a correlation between BVIs. Thus, for example, it is possible to create a Category-Manufacturer BVI that includes BVs that correlate each of the categories with each of the manufacturers. Thus, examination of one of the BVs in the Category-Manufacturer BVI identifies which manufacturers sell what product category (or categories). Similarly, it is possible to create a Manufacturer-to-Category BVI to identify which product category (or categories) are offered by a particular manufacturer.

Fig. 10 provides an example of Category-Manufacturer and Manufacturer-Category BVIs according to the present invention. Each BV in BVI 1000 identifies the manufacturers that offer a given product category. A bit in a BV of BVI 1000 correlates a category with a manufacturer. For example, bit 1012 correlates the "Apex" manufacturer with the "Computer" category. Similarly, each BV in BVI 1001 identifies the product categories offered by a given manufacturer. Each bit (e.g., bit 1013) correlates a manufacturer (e.g., "Apex") and a category (e.g., "Computer").

As discussed above, a BV that reflects occurrences of values in an indexed table such as products table 300 is updated to reflect a change made to a record in the indexed table (e.g., products table 300). A change in a BV of a retained value-limiting BVI may also be necessary.

For example, assume that the Category ID in record 312 is changed from "2" to "1". BVs 340 and 341 are updated as well to reflect the change as is illustrated in BVs 1340 and 1341, respectively. The inquiry then becomes whether or not BVIs 1000 and 1001 should also be updated. The third bit of BV 1340 is updated to indicate that an additional record of products table 300 refers to Category ID="1". Similarly, the third bit of BV 1341 is updated to reflect that record 312 in products table 300 refers has been changed to Category ID="1".

The focus of inquiry with respect to value-limiting BVIs 1000 and 1001 is whether or not the intersections of Category ID="2" and Manufacturer ID="2" need to be updated. Bit 1014 of BVI 1000 and bit 1015 of BVI 1001 should reflect the newly-created relationship while bit 1012 of BVI 1000 and 1013 of BVI 1001 of BVI 1001 reflect the relationship severed by the update. Updating these bits is based on whether there are other records in products table 300 that include both a Category ID and a Manufacturer ID equal to "2". Instead of recreating BVIs 1000 and 1001, it is possible to update only those BVs that are effected by the update (i.e., those BVs involving Categories "1" and "2" and Manufacturer "2").

Bits 1014 and 1015 correspond to a combination of Category "1" and Manufacturer "2", and the change to products table 300 created a relationship between these values. If another record contains this relationship, bits 1014 and 1015 are already set. However, bits 1014 and 1015 are unset indicating that there is no prior relationship between these values of Category and Manufacturer. Bits 1014 and 1015 are therefore set to reflect the newly-created relationship.

Bits 1014 and 1015 may simply be set. Alternatively, a logical "AND" may be performed between the updated BV 340 (i.e., BV 1340, "10110") and BV 331 (i.e., "00101"). The result of the logical "AND" reflects the fact that the update created a combination of Manufacturer and Category that did not previously exist. Bits 1014 and 1015 are updated to reflect the new combination.

In contrast to bits 1014 and 1015 that involve the relationship created by the update, bits 1012 and 1013 involve the relationship severed by the update. What is not immediately apparent is whether there is another record in products table 300 that contains the same relationship (i.e., Category "2" and Manufacturer "2"). If not, bits 1012 and 1013 need to be reset. If there is another relationship, however, there is no need to update bits 1012 and 1013.

The simple (but inefficient) approach is to simply run through the records to see if any records match. An alternative approach is to first update BVI 304 (i.e., BVs 340 and 341 are updated as BVs 1340 and 1341) to reflect the update to products table 300. Once BVI 304 is updated, BV 1341 (i.e., "01000") from BVI 304 is logical "ANDed" with BV 331 ("00101") of BVI 303 to determine whether any records meet both the Category "2" and Manufacturer "2" criteria after the update. If the result is "TRUE", there is no need to update bits 1012 and 1013. If the result is "FALSE", bits 1012 and 1013 are unset.

In the value-limiting examples discussed above, an "AND" operation is performed to limit one value set by another value set. Value-limiting may be used to limit the value selections made available to a user. Thus, given a particular value for Manufacturer (e.g., "ACME"), value-limiting may be used to identify those Category values that are associated in one or more records of the products table 300. In addition to "AND", it is possible to "OR" BVs to identify existing value combinations or lack thereof.

The present invention performs bit-level operations such as the bit-wise and logical operations described above. It is possible to further optimize these operations by storing the number of bits set in a BV and the position of the first set bit. This provides the ability to start an operation at the first set bit rather than the beginning of a BV. It is also possible to stop after all the set bits have been encountered rather than traversing to the end of a BV. Where such information is known for both BV operands used in an operation, the operation between the BVs may start at the earliest set bit and stop at the last set bit known for the BVs.

In this regard, the invention has been described with respect to particular illustrative embodiments. However, it is to be understood that the invention is not limited to the above-described embodiments and that various changes and modifications may be made by those of ordinary skill in the art without departing from the spirit and the scope of the invention.

WHAT IS CLAIMED IS:

1. A method of indexing occurrences of a value in at least one data record using a bit vector representation, the method comprising:

associating a bit vector representation with a value;

associating a bit position of the bit vector representation to the at least one record;

determining whether the value exists in the at least one data record; and

assigning a value to the bit position in the bit vector representation based on the outcome of the determining step;

synchronizing the bit position with the value to reflect any updates to the value.

2. A method according to Claim 1, further comprising:

encoding the bit vector representation.

3. A method according to Claim 2, wherein the bit vector representation comprises a sequence of bits, encoding the bit vector representation further comprising:

determining whether a frequency of a binary digit is less than a threshold value; and storing as the encoded bit vector representation at least one position of the binary digit in the bit vector representation.

4. A method according to Claim 3, wherein the threshold is the number of bits used to store a number.

5. A method according to Claim 2, wherein the bit vector representation comprises a sequence of binary digits, encoding the bit vector representation further comprising:

determining whether a size of a region of like binary digits is greater than a threshold value; and

storing as the encoded bit vector representation a representation of the region.

6. A method according to claim 5, wherein the representation comprises a start and end designation pair that represents the start and ending bits of the region.

7. A method according to Claim 5, wherein the threshold is twice the number of bits used to store a number.

8. A method according to Claim 1, further comprising:

compressing the encoded bit vector representation.

9. A method according to Claim 1, wherein the bit vector representation is compressed using a compression technique.

10. A method according to Claim 1, wherein the bit vector representation is encoded and compressed.

11. A method according to Claim 1, wherein the data structure is a record in a database.

12. A method according to Claim 1, further comprising:

examining the bit vector representation to determine whether the data record contains the value.

13. A method according to Claim 1, wherein plural bit vector representations exist each representing a discrete value, the method further comprising:

determining whether the data record contains more than one of the values by performing a bit-level operation on the corresponding bit vector representations.

14. A method according to Claim 9, wherein the bit-level operation is an "OR" operation.

15. A method according to Claim 13, wherein the operation is an "AND" operation.

16. A method of identifying combinations of values used in at least one data record comprising fields for storing the values, the method comprising:

creating a first bit vector representation for a first value, the first bit vector representation identifying use of the first value in the at least one data record;

creating a second bit vector representation for a second value, the second bit vector representation identifying use of the second value in the at least one data record; and

performing a bit-level operation on the first and second bit vector representations.

17. A method according to Claim 16, wherein the bit-level operation is an "AND" operation.

18. A method according to Claim 17, wherein the "AND" operation is a bit-wise "AND" returning a bit corresponding to each of the at least one data record identifying whether a combination of the first and second values exist in the at least one data record.

19. A method according to Claim 17, wherein the "AND" operation is a logical "AND" returning a single result representing whether any of the at least one data record contains a combination of the first and second values.

20. A method according to Claim 16, further comprising:

updating the at least one data record.

21. A method according to Claim 20, wherein the update to the at least one data record changes the first value, the method further comprising:

updating the first bit vector representation to reflect the update to the at least one data record.

22. A computer-readable memory medium in which computer-executable process steps are stored, the process steps for indexing occurrences of a value in at least one data record using a bit vector representation, wherein the process steps comprise:

an associating step to associate a bit vector representation with a value;

an associating step to associate a bit position of the bit vector representation to the at least one record;

a determining step to determine whether the value exists in the at least one data record; and

an assigning step to assign a value to the bit position in the bit vector representation based on the outcome of the determining step;

a synchronizing step to synchronize the bit position with the value to reflect any updates to the value.

23. A computer-readable memory medium according to Claim 22, further comprising:

an encoding step to encode the bit vector representation.

24. A computer-readable memory medium according to Claim 23, wherein the bit vector representation comprises a sequence of bits,

encoding the bit vector representation further comprising:

a determining step to determine whether a frequency of a binary digit is less than a threshold value; and

a storing step to store as the encoded bit vector representation at least one position of the binary digit in the bit vector representation.

25. A computer-readable memory medium according to Claim 24, wherein the threshold is the number of bits used to store a number.

26. A computer-readable memory medium according to Claim 23, wherein the bit vector representation comprises a sequence of binary digits, encoding the bit vector representation further comprising:

a determining step to determine whether a size of a region of like binary digits is greater than a threshold value; and

a storing step to store as the encoded bit vector representation a representation of the region.

27. A computer-readable memory medium according to claim 26, wherein the representation

comprises a start and end designation pair that represents the start and ending bits of the region.

28. A computer-readable memory medium according to Claim 26, wherein the threshold is twice the number of bits used to store a number.

29. A computer-readable memory medium according to Claim 22, further comprising:

a compressing step to compress the encoded bit vector representation.

30. A computer-readable memory medium according to Claim 22, wherein the bit vector representation is compressed using a compression technique.

31. A computer-readable memory medium according to Claim 22, wherein the bit vector representation is encoded and compressed.

32. A computer-readable memory medium according to Claim 22, wherein the data structure is a record in a database.

33. A computer-readable memory medium according to Claim 22, further comprising:

an examining step to examine the bit vector representation to determine whether the data record contains the value.

34. A computer-readable memory medium according to Claim 22, wherein plural bit vector representations exist each representing a discrete value, the method further comprising:

a determining step to determine whether the data record contains more than one of the values by performing a bit-level operation on the corresponding bit vector representations.

35. A computer-readable memory medium according to Claim 30, wherein the bit-level operation is an "OR" operation.

36. A computer-readable memory medium according to Claim 34, wherein the operation is an "AND" operation.

37. A computer-readable memory medium in which computer-executable process steps are stored, the process steps for identifying combinations of values used in at least one data record comprising fields for storing the values, wherein the process steps comprise:

a creating step to create a first bit vector representation for a first value, the first bit vector representation identifying use of the first value in the at least one data record;

a creating step to create a second bit vector representation for a second value, the second bit vector representation identifying use of the second value in the at least one data record; and

a performing step to perform a bit-level operation on the first and second bit vector representations.

38. A computer-readable memory medium according to Claim 37, wherein the bit-level operation is an "AND" operation.

39. A computer-readable memory medium according to Claim 38, wherein the "AND" operation is a logical "AND" returning a bit corresponding to each of the at least one data record identifying whether a combination of the first and second values exist in the at least one data record.

40. A computer-readable memory medium according to Claim 38, wherein the "AND" operation is a bit-wise "AND" returning a single result representing whether any of the at least one data record contains a combination of the first and second values.

41. A computer-readable memory medium according to Claim 37, further comprising:

an updating step to update the at least one data record.

42. A computer-readable memory medium according to Claim 41, further comprising:

a determining step to determine whether the update to the at least one data record effects the first bit vector representation;

an updating step to update the first bit vector representation, if it is determined that the update to the at least one data record effects the first bit vector representation;

a determining step to determine whether the update to the at least one data record effects the second bit vector representation; and

an updating step to update the second bit vector representation, if it is determined that the update to the at least one data record effects the second bit vector representation.

43. Computer-executable process steps stored on a computer readable medium, said computer-executable process steps for indexing occurrences of a value in at least one data record using a bit vector representation, said computer-executable process steps comprising:

code to associate a bit vector representation with a value;

code to associate a bit position of the bit vector representation to the at least one record;

code to determine whether the value exists in the at least one data record; and

code to assign a value to the bit position in the bit vector representation based on the outcome of the determining step;

code to synchronize the bit position with the value to reflect any updates to the value.

44. A computer-executable process steps according to Claim 43, further comprising:

code to encode the bit vector representation.

45. A computer-executable process steps according to Claim 44, wherein the bit vector representation comprises a sequence of bits, encoding the bit vector representation further comprising:

code to determine whether a frequency of a binary digit is less than a threshold value; and

code to store as the encoded bit vector representation at least one position of the binary digit in the bit vector representation.

46. A computer-executable process steps according to Claim 45, wherein the threshold is the number of bits used to store a number.

47. A computer-executable process steps according to Claim 44, wherein the bit vector representation comprises a sequence of binary digits, encoding the bit vector representation further comprising:

code to determine whether a size of a region of like binary digits is greater than a threshold value; and

code to store as the encoded bit vector representation a representation of the region.

48. A computer-executable process steps according to claim 50, wherein the representation comprises a start and end designation pair that represents the start and ending bits of the region.

49. A computer-executable process steps according to Claim 47, wherein the threshold is twice the number of bits used to store a number.

50. A computer-executable process steps according to Claim 43, further comprising:

code to compress the encoded bit vector representation.

51. A computer-executable process steps according to Claim 43, wherein the bit vector representation is compressed using a compression technique.

52. A computer-executable process steps according to Claim 43, wherein the bit vector representation is encoded and compressed.

53. A computer-executable process steps according to Claim 43, wherein the data structure is a record in a database.

54. A computer-executable process steps according to Claim 43, further comprising:

code to examine the bit vector representation to determine whether the data record contains the value.

55. A computer-executable process steps according to Claim 43, wherein plural bit vector representations exist each representing a discrete value, the method further comprising:

code to determine whether the data record contains more than one of the values by performing a bit-level operation on the corresponding bit vector representations.

56. A computer-executable process steps according to Claim 51, wherein the bit-level operation is an "OR" operation.

57. A computer-executable process steps according to Claim 55, wherein the operation is an "AND" operation.

58. Computer-executable process steps stored on a computer readable medium, said computer-executable process steps for identifying combinations of values used in at least one data record comprising fields for storing the values, said computer-executable process steps comprising:

code to create a first bit vector representation for a first value, the first bit vector representation identifying use of the first value in the at least one data record;

code to create a second bit vector representation for a second value, the second bit vector representation identifying use of the second value in the at least one data record; and

code to perform a bit-level operation on the first and second bit vector representations.

59. A computer-executable process steps according to Claim 58, wherein the bit-level operation is an "AND" operation.

60. A computer-executable process steps according to Claim 59, wherein the "AND" operation is a logical "AND" returning a bit corresponding to each of the at least one data record identifying whether a combination of the first and second values exist in the at least one data record.

61. A computer-executable process steps according to Claim 59, wherein the "AND" operation is a bit-wise "AND" returning a single result representing whether any of the at least one data record contains a combination of the first and second values.

62. A computer-executable process steps according to Claim 58, further comprising:

code to update the at least one data record.

63. A computer-executable process steps according to Claim 62, further comprising:

code to determine whether the update to the at least one data record effects the first bit vector representation;

code to update the first bit vector representation, if it is determined that the update to the at least one data record effects the first bit vector representation;

code to determine whether the update to the at least one data record effects the second bit vector representation; and

code to update the second bit vector representation, if it is determined that the update to the at least one data record effects the second bit vector representation.

ABSTRACT

The present invention provides for indexing of occurrences of a value in at least one data record using a bit vector wherein a bit vector is associated with the value and a bit of the bit vector representation is associated with the at least one data record, a determination is made whether the value exists in the at least one data record, a bit value is assigned to the bit in the bit vector representation based on the outcome of the determination. Further, operations may be performed on multiple bit vectors indexing data records and values used in the data records to determine the existence of combinations and associations between the corresponding values and the indexed data records.

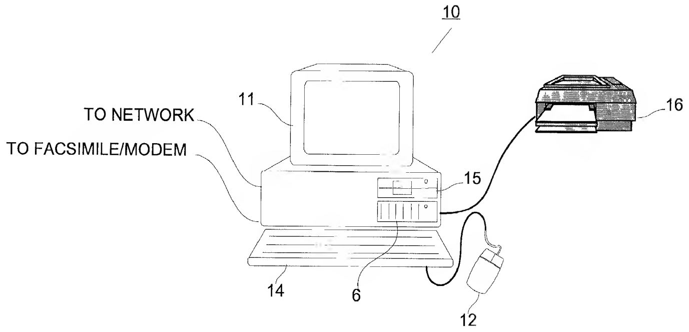


Fig. 1

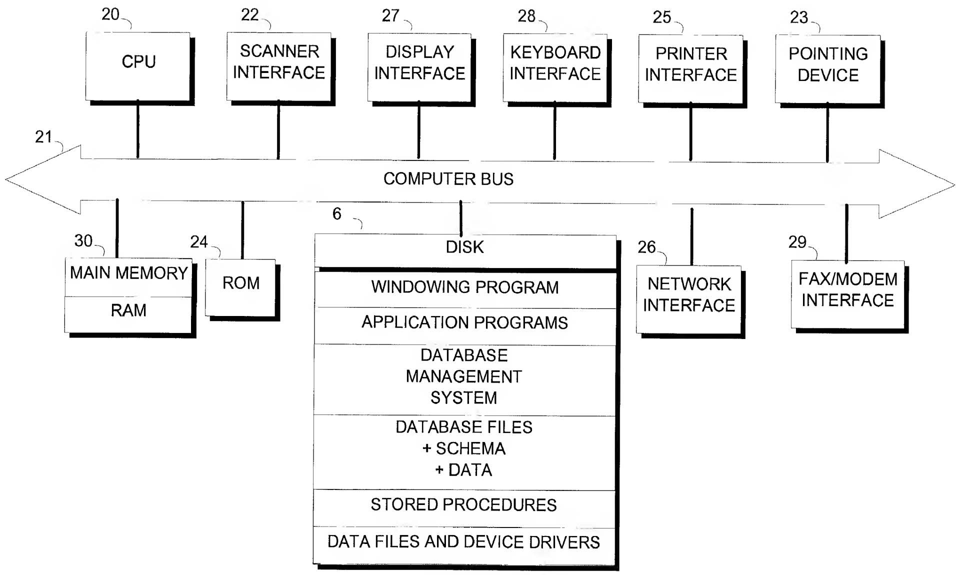
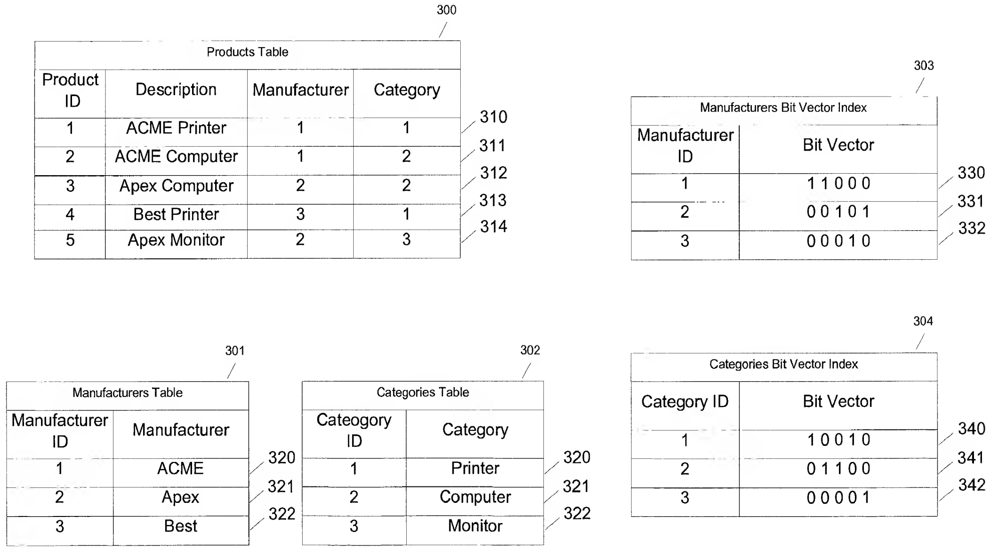

20
CPU
22
SCANNER INTERFACE
27
DISPLAY INTERFACE
28
KEYBOARD INTERFACE
25
PRINTER INTERFACE
23
POINTING DEVICE
21
COMPUTER BUS
30
MAIN MEMORY
RAM
24
ROM
6
DISK
WINDOWING PROGRAM
APPLICATION PROGRAMS
DATABASE MANAGEMENT SYSTEM
DATABASE FILES + SCHEMA + DATA
STORED PROCEDURES
DATA FILES AND DEVICE DRIVERS
26
NETWORK INTERFACE
29
FAX/MODEM INTERFACE

Fig. 2



300
301
302
303
304
310
311
312
313
314
320
321
322
330
331
332
340
341
342

Products Table

Product ID	Description	Manufacturer	Category
1	ACME Printer	1	1
2	ACME Computer	1	2
3	Apex Computer	2	2
4	Best Printer	3	1
5	Apex Monitor	2	3

Manufacturers Bit Vector Index

Manufacturer ID	Bit Vector
1	1 1 0 0 0
2	0 0 1 0 1
3	0 0 0 1 0

Manufacturers Table

Manufacturer ID	Manufacturer
1	ACME
2	Apex
3	Best

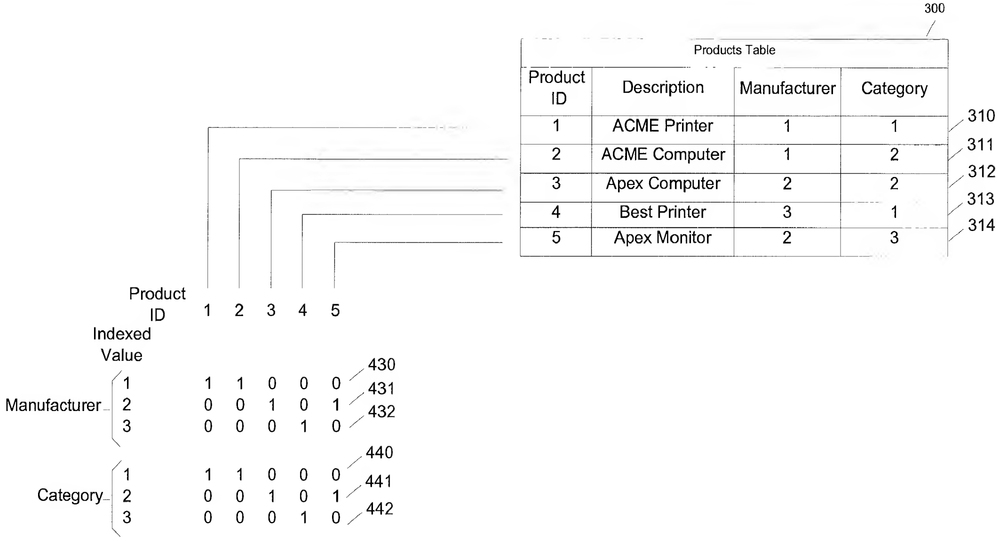
Categories Table

Cateogory ID	Category
1	Printer
2	Computer
3	Monitor

Categories Bit Vector Index

Category ID	Bit Vector
1	1 0 0 1 0
2	0 1 1 0 0
3	0 0 0 0 1

Fig. 3



Products Table

Product ID	Description	Manufacturer	Category
1	ACME Printer	1	1
2	ACME Computer	1	2
3	Apex Computer	2	2
4	Best Printer	3	1
5	Apex Monitor	2	3

	Indexed Value	Product ID 1	2	3	4	5
Manufacturer	1	1	1	0	0	0
	2	0	0	1	0	1
	3	0	0	0	1	0
Category	1	1	1	0	0	0
	2	0	0	1	0	1
	3	0	0	0	1	0

Fig. 4

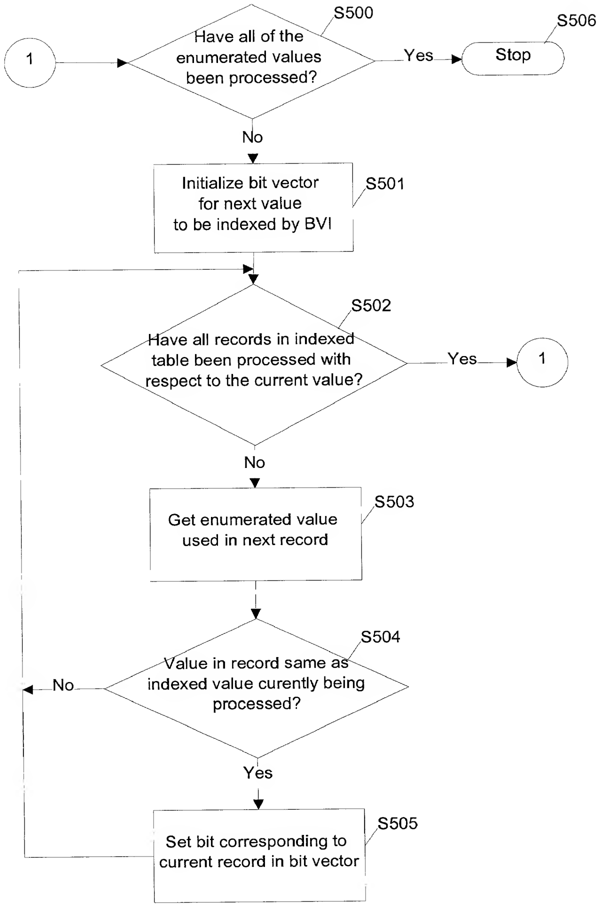

1
Have all of the enumerated values been processed?
S500
Yes
Stop
S506
No
Initialize bit vector for next value to be indexed by BVI
S501
Have all records in indexed table been processed with respect to the current value?
S502
Yes
1
No
Get enumerated value used in next record
S503
Value in record same as indexed value curently being processed?
S504
No
Yes
Set bit corresponding to current record in bit vector
S505

Fig. 5

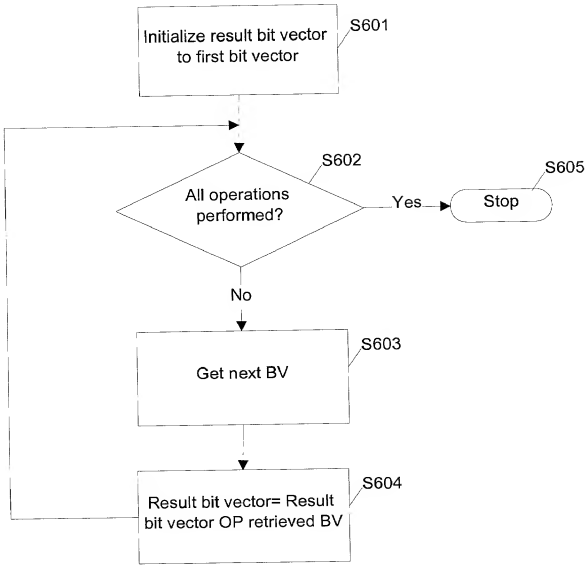
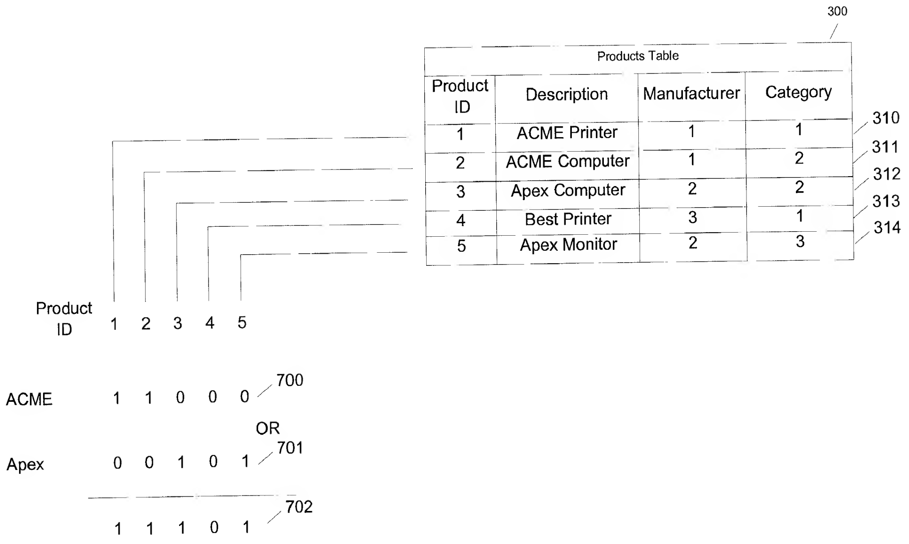

Initialize result bit vector to first bit vector
S601
S602
All operations performed?
Yes
S605
Stop
No
S603
Get next BV
S604
Result bit vector= Result bit vector OP retrieved BV

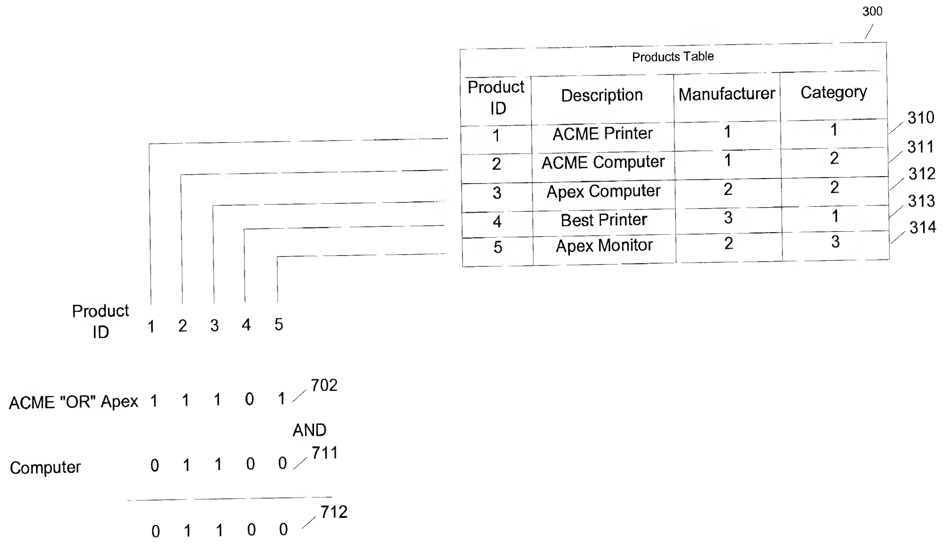
Fig. 6


Products Table

Product ID	Description	Manufacturer	Category
1	ACME Printer	1	1
2	ACME Computer	1	2
3	Apex Computer	2	2
4	Best Printer	3	1
5	Apex Monitor	2	3

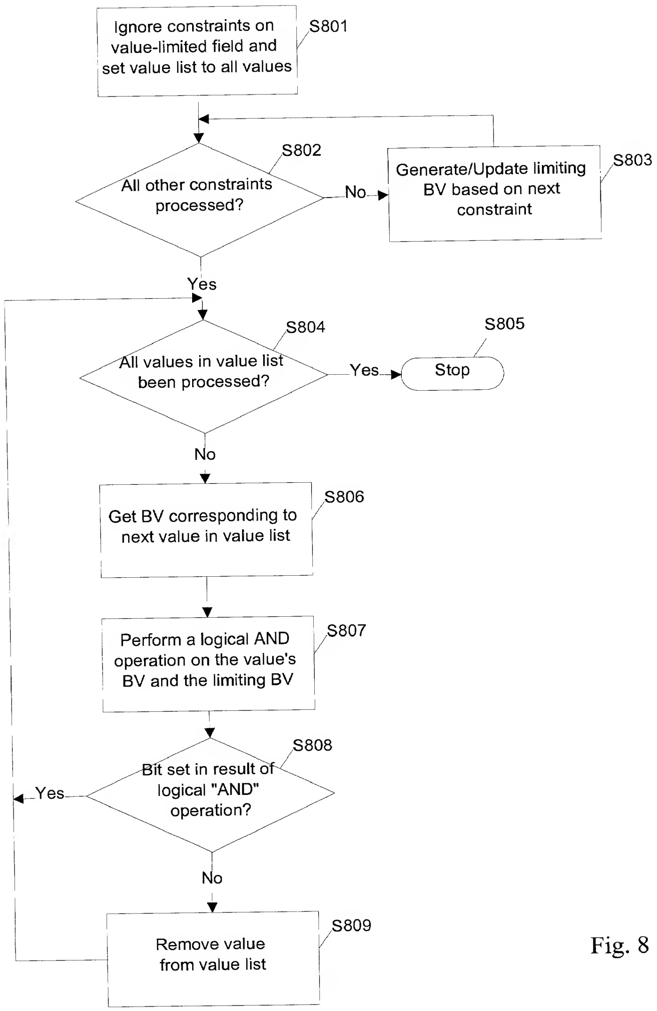
Product ID	1	2	3	4	5
ACME	1	1	0	0	0
Apex	0	0	1	0	1
	1	1	1	0	1

Fig. 7A



Product ID	Description	Manufacturer	Category
1	ACME Printer	1	1
2	ACME Computer	1	2
3	Apex Computer	2	2
4	Best Printer	3	1
5	Apex Monitor	2	3

Fig. 7B



Ignore constraints on value-limited field and set value list to all values
S801
All other constraints processed?
S802
No
Generate/Update limiting BV based on next constraint
S803
Yes
All values in value list been processed?
S804
Yes
Stop
S805
No
Get BV corresponding to next value in value list
S806
Perform a logical AND operation on the value's BV and the limiting BV
S807
Bit set in result of logical "AND" operation?
S808
Yes
No
Remove value from value list
S809

Fig. 8

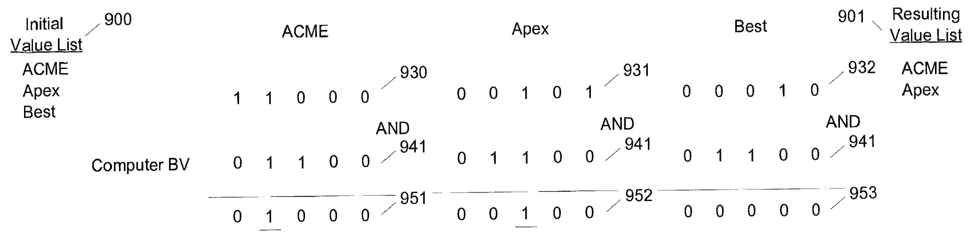

Initial
Value List
900
ACME
Apex
Best
ACME
Apex
Best
901
Resulting
Value List
ACME
Apex
1 1 0 0 0
930
0 0 1 0 1
931
0 0 0 1 0
932
AND
Computer BV
0 1 1 0 0
941
0 1 0 0 0
951
0 0 1 0 0
952
0 0 0 0 0
953

Fig. 9A

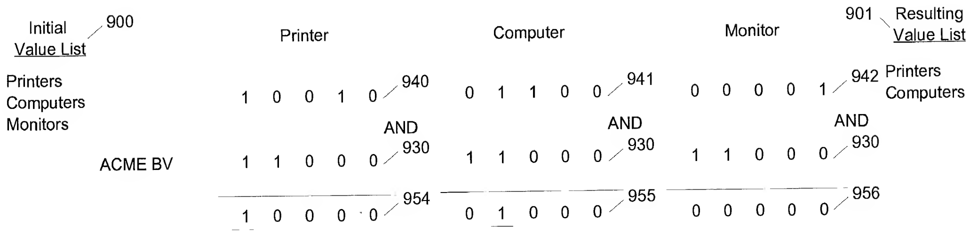

Initial
Value List
900
Printers
Computers
Monitors
Printer
Computer
Monitor
901
Resulting
Value List
Printers
Computers
1 0 0 1 0
940
0 1 1 0 0
941
0 0 0 0 1
942
AND
ACME BV
1 1 0 0 0
930
1 0 0 0 0
954
0 1 0 0 0
955
0 0 0 0 0
956

Fig. 9B

1000

CategoryID-to-ManufacturerID Bit Vector Index

Manufacturer ID	Bit Vector		
1	1	1014 0	1
2	1	1012 1	0
3	0	1	0

1001

ManufacturerID-to-CategoryID Bit Vector Index

Category ID	Bit Vector		
1	1	1	0
2	1015 0	1013 1	1
3	1	0	0

Category ID = 1	1	0	1	1	0	1340
Cagegory ID = 2	0	1	0	0	0	1341

Fig. 10